МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Линейные структуры данных: стек, очередь

Вариант 10

Студент гр. 8304 ___ Нам Ё Себ

Преподаватель ___ Фиалковский М. С.

Санкт-Петербург

2019

Цель работы.

Получить опыт работы с очередью и стеком (ссылочная реализация).

Постановка задачи.

Определить, имеет ли заданная в файле F символьная строка следующую структуру: a D b D c D d ..., где каждая строка a, b, c, d, ..., в свою очередь, имеет вид x1 C x2. Здесь x1 есть строка, состоящая из символов A и B, а x2 - строка, обратная строке x1 (т. е. если x1 = ABABB, то x2 = BBABA). Таким образом, исходная строка состоит только из символов A, B, C и D. Исходная строка может читаться только последовательно (посимвольно) слева направо.

Описание алгоритма.

1) Считывание осуществляется при помощи конструкции:

```
while (std::getline(in, currentFileString))
```

2) После считывания очередной строки, подстрока x1 посимвольно кладется на стек. Далее подстрока x2 посимвольно сравнивается с ранее сохраненной подстрокой x1, так как для решения поставленной подзадачи был использован стек, то подстрока x1 хранится в нем перевернутой.

Спецификация программы.

Программа предназначена для проверки строк, хранящихся в файле F, и записи результата проверки в файл G. Программа написана на языке C++.

Описание функций и структур данных.

1) Для хранения значений в стеке был реализован шаблонный класс Stack. Данный класс содержит 2 поля – поле size_, соответствующее кол-ву эл-ов в стеке и поле head_, соответствующее текущему верхнему эл-ту стека. Каждый элемент стека представляет собой объект класса Node, имеющего 2 поля – поле next, соответствующее следующего эл-ту стека, и поле data_, хранящее значение

текущего эл-та.

2) Для решения поставленной подзадачи была реализована функция check, которая осуществляет посимвольную проверку очередной строки считанной из входного файла. Тип возвращаемого значения – bool, что позволяет головной функции определить результат проверки и записать соответствующее значение в выходной файл.

Тестирование.

Таблица 1 – Результаты тестирования программы

Содержимое файла F	**Содержимое файла G (после запуска программы)**
ABCBA	YES
ABCBADABCBADBBCBBDBACAB	YES
C	NO
EGBEIGNWIHHGOWIGHOWGbgjkkGBKjgebh83	Uncorrect symbols
())()()()	Uncorrect symbols
ABCAB	NO
ACA	YES
AAAAAABCBAAAAAADAACAADBABCBABDBBBCBBB	YES
ABDBA	NO
ABCCCCBA	NO
D	NO

Выводы.

В ходе работы был получен опыт работы со стеком (ссылочная реализация). Исходный код программы представлен в приложении А.

Приложение А. Исходный код программы.

Main.cpp

```
#include "my_stack.h"

void readInputData(std::ifstream& in, std::vector<std::string>& data)
{
	std::string tmp;
	while (std::getline(in, tmp))
	{
		if (tmp.back() == '\r')
			tmp.erase(tmp.end() - 1);
		data.push_back(tmp);
	}
}

bool checkSymbolsCorection(std::string const& processedString)
{
	for (char c : processedString)
	{
		if (c < 'A' || c > 'D')
			return false;
	}

	return true;
}

bool check(std::string const& line)
{
	Stack<char> st;

	for (int i = 0; i < line.length(); ++i)
	{
		while (line[i] != 'C')
		{
			st.push(line[i]);

			++i;
			if (i == line.length())
				return false;
		}

		++i;
		while (line[i] != 'D')
		{
			if (st.empty())
				return false;

			char c = st.front();
			st.pop();

			if (c != line[i])
				return false;

			++i;
```

```
                if (i == line.length())
                        return st.empty();
            }
        }

        return true;
}

int main(int argc, char** argv)
{
        if (argc > 2)
        {
            std::ifstream in(argv[1]);
            if (!in)
            {
                std::cout << "Uncorrect input file\n";
                return 0;
            }
            std::ofstream out(argv[2]);
            if (!out)
            {
                std::cout << "Uncorrect output file\n";
                return 0;
            }

            std::vector<std::string> inputData;
            readInputData(in, inputData);

            for (auto const& line : inputData)
            {
                bool symbolsCheckingResult = checkSymbolsCorection(line);
                if (symbolsCheckingResult == false)
                {
                    out << "Uncorrect symbols\n";
                    continue;
                }

                bool controlCheckResult = check(line);
                out << ((controlCheckResult) ? "YES\n" : "NO\n");
            }
        }

        return 0;
}
}
}
```

My_stach.h

```
#pragma once
#include <memory>
#include <iostream>
#include <fstream>
#include <vector>
#include <string>

template <typename T>
class Stack
{
public:
	struct Node;
	using NodePtr = std::shared_ptr<Node>;

	struct Node
	{
		Node() = default;

		T data = T();
		NodePtr next = nullptr;
	};

	Stack() = default;
	Stack(int new_size, int arg) : size_(new_size)
	{
		for (int i = 0; i < size_; ++i)
			push(arg);
	}

	Stack(Stack const& other)
	{
		head_ = other.head_;
		size_ = other.size_;
	}

	Stack& operator=(Stack const& other)
	{
		head_ = other.head_;
		size_ = other.size_;

		return *this;
	}

	bool empty()
	{
		return size_ == 0;
	}

	size_t size()
	{
		return size_;
	}

	T front()
	{
		return head_->data;
	}

	void push(T arg)
	{
		auto newElem = std::make_shared<Node>();
```

```
            newElem->next = head_;
            newElem->data = arg;
            head_ = newElem;

            ++size_;
      }

      void pop()
      {
            if (empty())
                  throw std::invalid_argument("Stack is empty");

            head_ = head_->next;

            --size_;
      }

private:
      size_t size_ = 0;
      NodePtr head_ = std::make_shared<Node>();
   };
```